VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

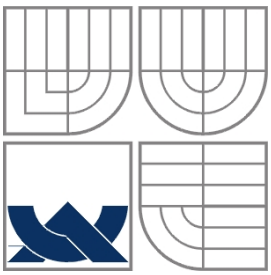
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

ZOBRAZENÍ MRAKŮ V REÁLNÉM ČASE

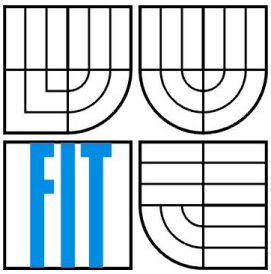
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE RADEK DOSTÁL
AUTHOR

BRNO 2009



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

ZOBRAZENÍ MRAKŮ V REÁLNÉM ČASE

RENDERING CLOUDS IN REAL TIME

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

RADEK DOSTÁL

VEDOUCÍ PRÁCE
SUPERVISOR

ING. ADAM HEROUT, PH.D.

BRNO 2009

Abstrakt

Práce se zabývá algoritmy schopnými zobrazit mraky v reálném čase. Teoretická část popisuje fyzikální princip oblaků a seznamuje s vybranými metodami pro jejich modelování a vykreslování. Cílem praktické části je implementovat jeden z algoritmů, schopný běžet v reálném čase a vyvinout aplikaci, která jej bude demonstrovat.

Abstract

This thesis is about algorithms which render clouds in real time. The theoretical section deals with clouds in real world and also describes some algorithms for modeling and rendering them. The aim of practical section is implement one of these real time algorithms and develop demonstrational application.

Klíčová slova

Mraky, OpenGL, GLEW, framebuffer object, Cg, celulární automaty, billboard, impostor, skybox, Perlinova šumová funkce

Keywords

Clouds, OpenGL, GLEW, framebuffer object, Cg, cellular automaton, billboard, impostor, skybox, Perlin noise

Citace

Radek Dostál: Zobrazení mraků v reálném čase, bakalářská práce, Brno, FIT VUT v Brně, 2009

Zobrazení mraků v reálném čase

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením Ing. Adama Herouta, Ph.D. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……
Radek Dostál
16.5. 2009

Poděkování

Děkuji vedoucímu mé práce Ing. Adamu Heroutovi, Ph.D. za odborné vedení, cenné rady a jeho čas, který mi věnoval při konzultacích.

© Radek Dostál, 2009
Tato práce vznikla jako školní dílo na Vysokém učení technickém v Brně, Fakultě informačních technologií. Práce je chráněna autorským zákonem a její užití bez udělení oprávnění autorem je nezákonné, s výjimkou zákonem definovaných případů.

Obsah

1 Úvod

Mraky jsou součástí takřka každé vizualizace exteriérů. Jejich nezakomponování do scény může mít fatální následky pro celkový dojem z realističnosti grafických aplikací. Počítačové hry (zejména letecké simulátory) by bez oblak asi nepůsobily příliš přesvědčivě a uživatele by nevtáhly do dění. Podobná situace může nastat u vizualizace v CAD systémech, kde si zákazník přeje vidět svůj výrobek v co nejreálnějších scénách.

Vzhledem k tomu, že mraky jsou z fyzikálního hlediska poměrně složité, o čemž stručně pojednává druhá kapitola, je třeba, zejména u aplikací běžících v reálném čase, provést co největší abstrakci, a přitom zachovat co nejvěrohodnější dojem.

Před samotným zobrazením je však třeba mrak vymodelovat, což popisuje třetí kapitola. Následuje kapitola sumarizující algoritmy zabývající se vykreslováním. Pátá kapitola pojednává o praktické implementaci vybraného algoritmu, použitých technologiích, optimalizacích a možných rozšířeních.

2 Mraky v reálném světě

2.1 Fyzikální podstata

Voda je v atmosféře zastoupena ve všech třech skupenstvích. Z aktivního povrchu se vypařuje, a tedy v plynném skupenství stoupá vzhůru. V důsledku klesání teploty společně s rostoucí nadmořskou výškou dochází ke kondenzaci páry na kapky. Vodní pára může také desublimovat, čímž vznikají ledové krystaly. Neboť kapky (popř. krystaly) padají velmi pomalu, stačí slabé výstupné proudy, aby docházelo k jejich shlukování. Takto nahromaděné kapky či krystaly tvoří mraky nebo poněkud odborněji oblaky. Konkrétní definice podle [RAC00] říká, že:

„Oblak je viditelná soustava nepatrných částic vody nebo ledu, popřípadě obojího, v ovzduší. Tato soustava může obsahovat zároveň i větší částice vody nebo ledu a také jiné částice pocházející např. z průmyslových exhalací, kouře nebo prachu.“

Podrobnější informace lze najít v publikaci [NET84].

2.2 Dělení oblak

Oblaka můžeme dělit pomocí několika kritérií. Z předchozí podkapitoly (2.1) vyplývá nejjednodušší způsob dělení, a to podle principu vzniku. Rozlišujeme pak oblaka:

1. vodní (vznikají z vodních kapek)
2. smíšená (tvořená směsí kapek a ledových krystalů)
3. ledová (tvořená ledovými krystaly)

Další možností, která je asi nejznámější, a i pro potřeby jejich zobrazování nejdůležitější, je dělení podle tvaru. Tato kategorizace zahrnuje 10 druhů oblak a využívá pro ně latinská označení:

1. řasa – Cirrus (Ci)
2. řasová kupa – Cirrocumulus (Cc)
3. řasová sloha – Cirrostratus (Cs)
4. vyvýšená kupa – Altocumulus (Ac)
5. vyvýšená sloha – Altostratus (As)
6. dešťová sloha – Nimbostratus (Ns)
7. slohová kupa – Stratocumulus (Sc)
8. sloha – Stratus (St)

9. kupa – Cumulus (Cu)
10. bouřkový mrak – Cumulonimbus (Cb)

Všechny tyto tvary mají mnoho odrůd, které se od sebe liší většinou jen nepatrně, takže jejich klasifikace může být, i pro odborníky, velmi obtížná.

Poslední dělení, které uvedu je dělení podle nadmořské výšky, ve které se vyskytují základny mraků. Jedná se o:

1. oblaka vysoká (Ci, Cc, Cs) - výška 5-13 km
2. oblaka střední (Ac, As) – výška od 2 do 7 km
3. oblaka nízká (Ns, Sc, St) – ve vyšce do 2 km
4. oblaka s vertikálním vývojem (Cu, Cb) – výška základny 0,5-1,5 km

Vysoká oblaka jsou složena výhradně z ledových krystalů, mají bílou barvu a jejich tvar může být vláknitý, mohou tvořit shluky ve tvaru chomáčků, či úzkých pruhů (tzv. beránky) nebo mohou tvořit souvislou bělavou vrstvu. Střední oblaka mají zpravidla bílou nebo slabě šedou barvu, často zakrývají celou oblohu a Slunce nimi většinou slabě prosvítá. Nízkými oblaky Slunce neprosvítá a mají tmavě šedou barvu.

2.3 Barva a jas

Kromě tvaru oblaku je pro zobrazení důležitá i jeho barva a jas. Jas je dán množstvím světla, které částice, z nichž se oblak skládá odrážejí. Zdrojem tohoto světla je nejčastěji Slunce nebo Měsíc. Jas může být také ovlivněn světelnými jevy (např. duha, koróna) nebo zákalem. Barva je ovlivněna barvou světla a polohou světelného zdroje od pozorovatele. Pokud je Slunce blízko obzoru, získávají oblaky žlutý až červený nádech. V případě, že je Slunce těsně nad obzorem může nastat situace, kdy vysoká oblaka mají bílou barvu, střední oblaka jsou červené a nízká oblaka šedé. Stejně jako jas mohou i barvu pozměnit světelné jevy či zákal.

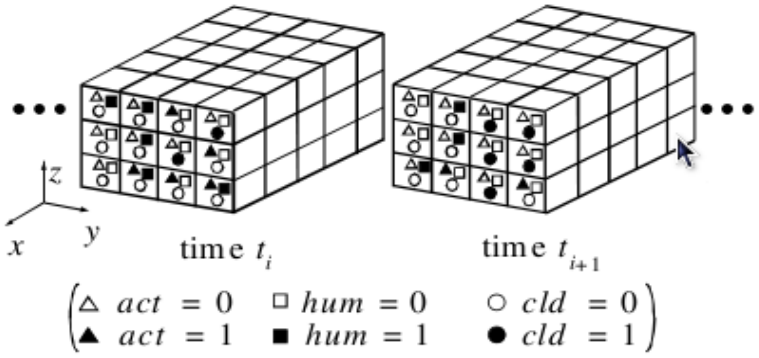
3 Modelování mraků

3.1 Realistické modelování

Nejpřirozenějším způsobem modelování mraků je snažit se napodobit přírodní děj. Vznik mraků je, jak je popsáno ve druhé kapitole, natolik složitý, že je v současné době velmi obtížné použít jeho přesný model pro grafické aplikace pracující v reálném čase. Pro potřeby takových aplikací je navíc poměrně zbytečné používat takto komplexní model a je vhodné šetřit zdroje počítače pro ostatní výpočty.

3.1.1 Modelování na bázi celulárního automatu

Pro vykreslování v reálném čase lze použít metodu popsanou v [DOB00]. Tento algoritmus zjednodušuje realistický přístup založený na fluidní dynamice a využívá celulárního (buněčného) automatu.



Obrázek 1: Znázornění přechodu ze stavu t_i na t_{i+1} celulárního automatu. Převzato z [DOB00]

Prostor je rozdělen na voxely, které odpovídají buňkám automatu. Ke každé buňce jsou přiřazeny tři dvoustavové proměnné (viz. Obrázek 1) označující vlhkost pro vznik oblaku (označena jako *hum*), přítomnost oblaku (*cld*) a aktivaci vzniku (*act*). Neboť se jedná o booleovské proměnné je možné definovat pravidla změny stavu buněčného automatu pomocí booleovských operací:

$$hum(i,j,k,t_{i+1}) = hum(i,j,k,t_i) \wedge \neg act(i,j,k,t_i)$$
$$cld(i,j,k,t_{i+1}) = cld(i,j,k,t_i) \vee act(i,j,k,t_i)$$
$$act(i,j,k,t_{i+1}) = \neg act(i,j,k,t_i) \wedge hum(i,j,k,t_i) \wedge f_{act}(i,j,k)$$

Proměnné *i*, *j* a *k* jsou indexy pole a proměnná t_i symbolizuje čas. Funkce f_{act} vrací hodnotu *PRAVDA* v případě, že je některá z okolních buněk aktivní:

$$f_{act} = act(i+1, j, k, t_i) \vee act(i, j+1, k, t_i) \vee act(i, j, k+1, t_i)$$
$$\vee act(i-1, j, k, t_i) \vee act(i, j-1, k, t_i) \vee act(i, j, k-1, t_i)$$
$$\vee act(i-2, j, k, t_i) \vee act(i+2, j, k, t_i) \vee act(i, j-2, k, t_i)$$
$$\vee act(i, j+2, k, t_i) \vee act(i, j, k-2, t_i)$$

Problém výše uvedených formulí tkví v tom, že po nastavení proměnné *cld* na *PRAVDA* zůstane v tomto stavu navždy. Situaci lze řešit pomocí generátoru náhodných čísel, který zajistí, že se oblak časem opět rozplyne.

Algoritmus je velmi zajímavým kompromisem mezi výsledkem a rychlostí.

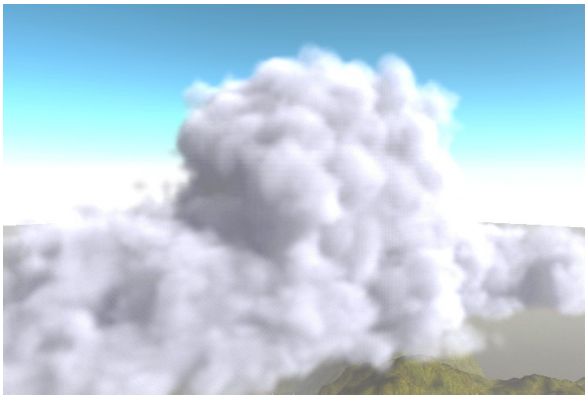
3.1.2 Fluidní dynamika

Realističtější způsob modelování oblak je založen na bázi atmosferické fluidní dynamiky. Tento algoritmus popsán v [MIY01] využívá metody CML (Coupled map lattice), což je rozšíření metody celulárního automatu. Rozdílem je, že CML využívá místo diskrétních proměnných, reálná čísla.

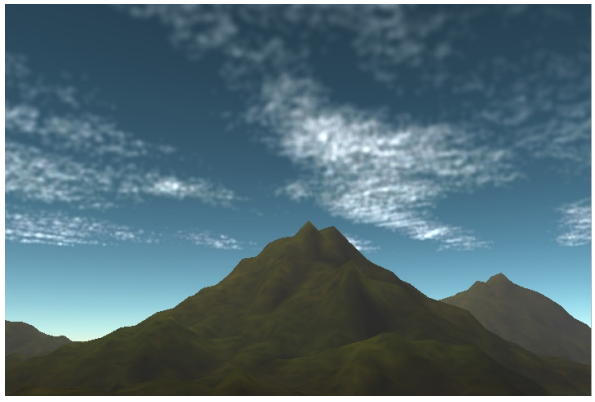
Autoři algoritmu vycházejí z Navier-Stokasovy rovnice:

$$\frac{D\boldsymbol{v}}{Dt} = \frac{-\nabla}{\rho} grad\ p + \nu \Delta \boldsymbol{v} \quad ,$$

kde $\boldsymbol{v}$ značí vektor rychlosti, ρ vyjadřuje hustotu, p tlak a konečně ν viskozitu. Tuto rovnici lze řešit numerickými metodami, které jsou ovšem velmi výpočetně náročné. Díky použití CML je metoda za cenu mírné abstrakce rychlejší, a na rozdíl od celulárního automatu dává možnost generovat mraky různých typů (viz. Obrázky 3 a 4). Také ji je možno implementovat hardwarově, v procesoru grafické karty.



Obrázek 2: Mrak typu Cumulonimbus generovaný metodou fluidní dynamiky. Převzato z [MIY01].



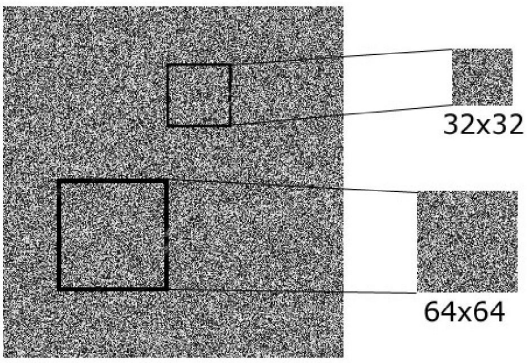
Obrázek 3: Mrak typu Cirrocumulus generovaný metodou fluidní dynamiky. Převzato z [MIY01].

3.2 Modelování pomocí náhodnosti

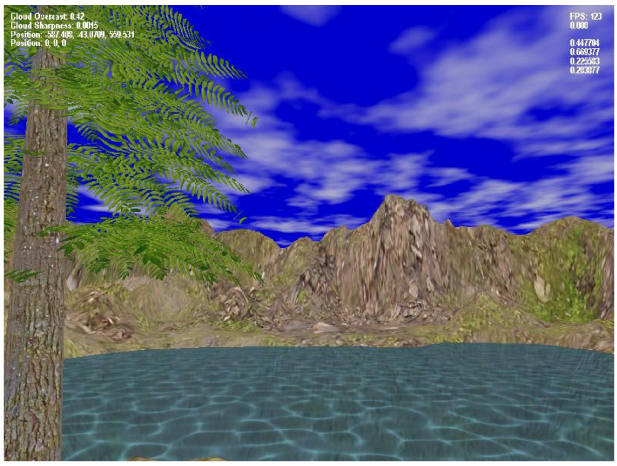
V případě, kdy realistické modelování není třeba, lze využít modelování pomocí generátoru pseudonáhodných čísel. Ke generování oblak se poté může použít fraktálních šumových funkcí (např. Perlinovy funkce). Další možností, která je blíže popsána v kapitole 5.2, je zformovat do oblaků otexturované poloprůhledné polygony a ty poté náhodně rozmístit do scény.

3.2.1 Využití Perlinovy funkce

Jedno z možných využití Perlinovy šumové funkce je popsáno v [HAS05]. Autoři využívají předpočítanou texturu použitou jako vyhledávací tabulku, o rozměrech 512x512 pixelů, obsahující šum. Z ní potom vyřezávají náhodné části a ty skládají ve výslednou texturu (viz. Obrázek 4). Takto získaná textura je nanesena na skybox, popř. skydome, o kterých bude pojednáno ve čtvrté kapitole. Výsledné oblaky (viz. Obrázek 5) nepůsobí fotorealisticky, ale protože je možné celý algoritmus počítat na procesoru grafické karty, je velmi rychlý i na starších počítačích (na grafické kartě ATI Radeon 9500Pro dosáhli autoři v průměru 60-100 FPS).



Obrázek 4: Vyhledávací tabulka vytvořena Perlinovou funkcí. Převzato z [HAS05].

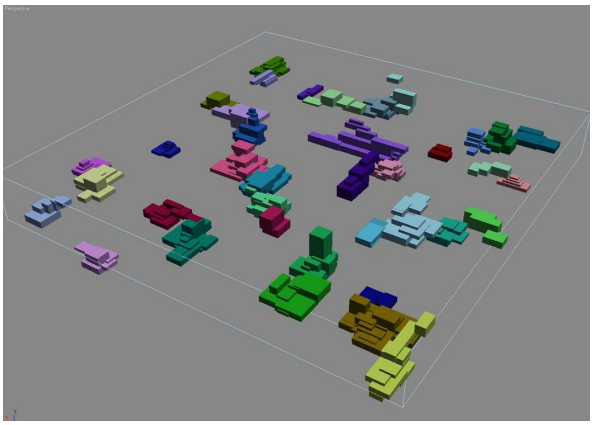


Obrázek 5: Výsledek použití Perlinovy funkce. Převzato z [HAS05].

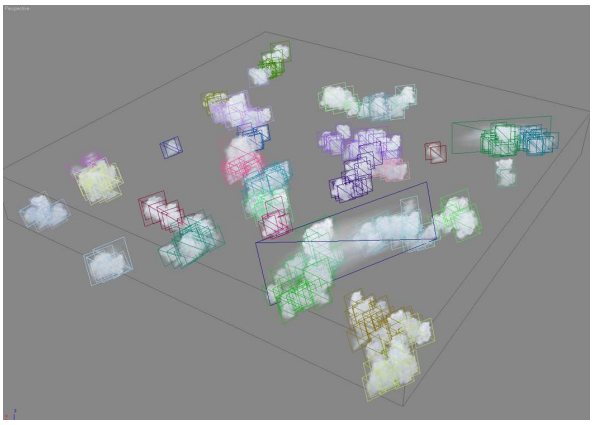
3.3 Předpřipravené

Velmi častý, a svým způsobem jednoduchý, způsob jak modelovat mraky je využít služeb některého modelovacího nástroje, výsledný model uložit do souboru a ten poté načíst do aplikace. Této varianty je využíváno především v počítačových hrách (např. Microsoft Flight Simulator 2004). Každý jednotlivý mrak je tedy pečlivě předpřipraven grafikem, což přináší možnost tvořit velmi pěkné

oblaky. Nevýhodou této metody je, že pokud je mraků vymodelováno málo, mohou se v aplikaci opakovat. Vhodným nástrojem pro tvorbu modelů může být například komerční Autodesk 3ds Max, do kterého je možné získat (např. z [TUT09]) rozšíření použité při vývoji výše zmíněné hry MS Flight Simulator. Oblak se v tomto případě sestaví z krychlí (viz. Obrázek 6), poté se pomocí skriptu převede na billboardy (viz. Obrázek 7).



Obrázek 6: Ukázka modelování mraků v Autodesk 3ds Max. Převzato z [WAN04]



Obrázek 7: Krychle skriptem nahrazeny za billboardy. Převzato z [WAN04]

4 Metody zobrazení mraků

Modely mraků získáme většinou jako volumetrická data nebo jako částicové systémy. Pro jednodušší implementace je možné reprezentovat mraky i pomocí dvourozměrné textury. V následujících podkapitolách jsou podrobněji popsány některé metody používané pro zobrazení takto získaných modelů, které je možné implementovat v aplikacích běžících v reálném čase. Podkapitola 4.4 se zabývá osvětlovacími modely oblaků.

4.1 Skybox / skydome

Velmi jednoduchý princip, kdy je scéna „obalena“ do kvádru, na jehož stěny jsou naneseny dvourozměrné textury oblohy. Základní varianta trpí několika neduhy. Prvním je nemožnost pohybu oblak, což může být vyřešeno nanesením procedurální textury (viz. 3.2.1), kterou je možné v průběhu času přepočítávat. Další problém vzniká v rozích kvádru, kde může být patrný zlom. Tato situace se obvykle řeší nahrazením krychle polokoulí (tzv. skydome). O průletech mraky samozřejmě nemůže být řeč, ale pro akční (použito v hrách Unreal, Half-Life 2 a mnohých dalších) či závodní hry je toto řešení velmi vhodné (viz. Obrázek 8).

Obrázek 8: Snímek obrazovky ze hry Half Life 2. Převzato z [GAM09].

4.2 Oblaky reprezentované obrázky a body

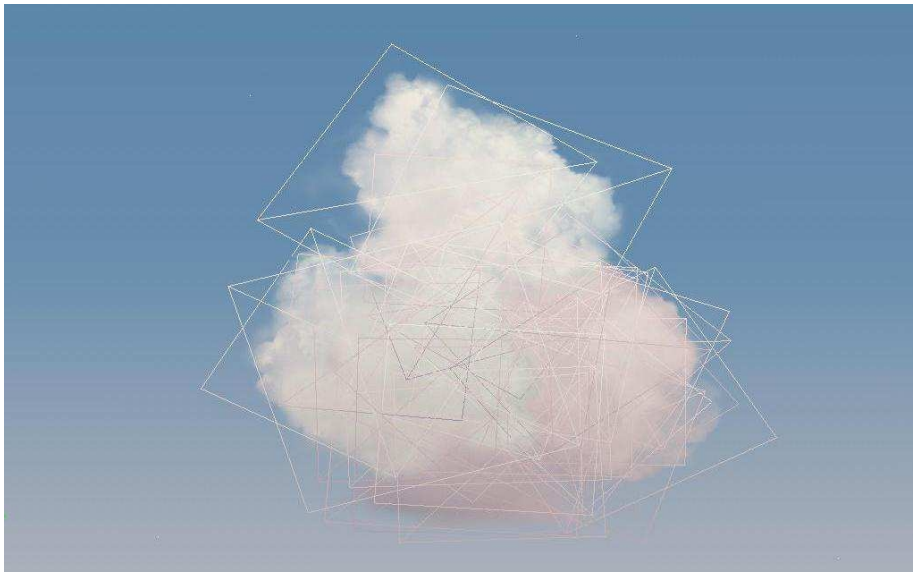
V aplikacích běžících v reálném čase nastává problém při zobrazování složitých modelů, jako jsou stromy, budovy či květiny. Tyto objekty se nejenom složitě modelují, ale jejich zobrazování je velmi výpočetně náročné. Z důvodů úspory výkonu se využívá techniky, kdy je model nahrazen jednoduchým polygonem, na který je nanesena textura zobrazovaného objektu. Podle použitých transformací a textury nazýváme polygony jako sprite, billboard nebo impostor.

Této metody je hojně využíváno i při vykreslování mraků, především pokud jsou mraky uloženy jako částicový systém, ale je ji možno aplikovat i na volumetrická data.

4.2.1 Sprite

Nejjednodušším případem je sprite. Jedná se o polygon, který je stále na stejném místě, a ani nanesená textura se v čase nemění. Problém spočívá pokud se pozorovatel dostane do místa, ze kterého je sprite vidět z boku. Tuto situaci lze řešit použitím dalšího polygonu se stejnou texturou, který bude na původní kolmý. Takovéto polygony se používají například u částicových systémů.

Právě sprite lze využít pro vykreslování oblak. Na každý polygon se nanese poloprůhledná textura zobrazující jednotlivé části mraku. Shluk takovýchto poloprůhledných polygonů natočených různým směrem vytvoří relativně realistický dojem mraku (viz. Obrázek 9). Pokud jsou polygony rozprostřeny ve všech osách v dostatečné hustotě, lze mrakem proletět. Tohoto způsobu využívá mj. algoritmus popsaný ve [WAN04].



Obrázek 9: Shluk většího množství polygonů tvořící mrak. Převzato z [WAN04].

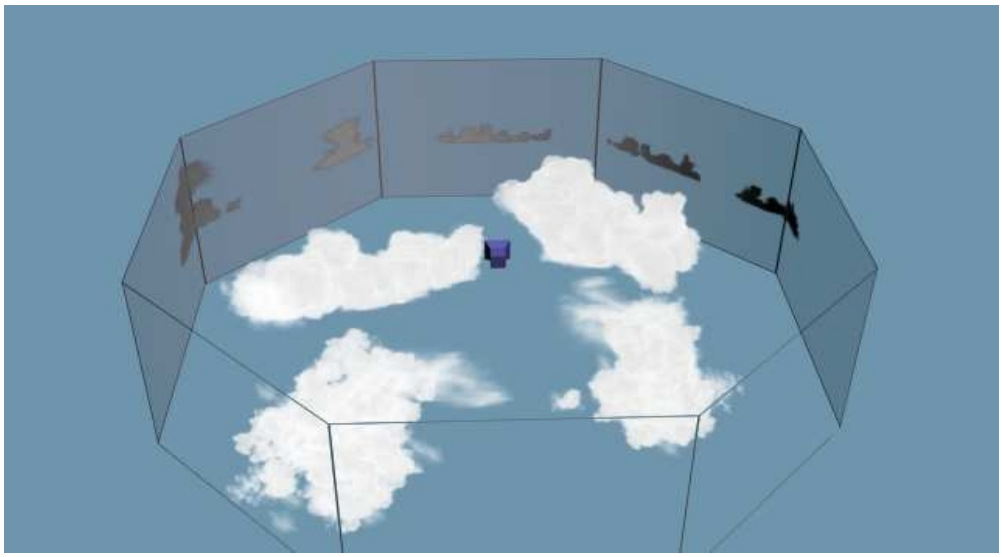
4.2.2 Billboard

Billboard je polygon, který se natáčí vždy směrem k pozorovateli. Tato rotace může být pouze podle osy *y*, čehož se využívá u osově souměrných objektů (např. stromy), nebo se může natáčet celý.

4.2.3 Impostor

Poslední možností, která nachází uplatnění ve všech moderních hrách je impostor. Ten se může chovat jako sprite nebo jako billboard, ale především jeho obsah je dynamicky obměňován. Díky tomu je možné na jeden polygon vykreslit více objektů a výrazně tak ušetřit výpočetní výkon. Této metody se využívá většinou v kombinaci s LOD (Level of Details), kdy vzdálené objekty se vykreslí jako impostory, a pokud se pozorovatel dostane do jeho blízkosti, je impostor zaměněn za plnohodnotný model.

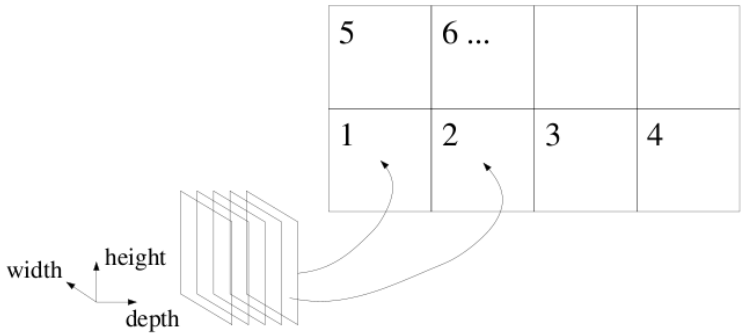
Impostorů se při zobrazování mraků využívá především v rámci optimalizací, kdy je do textury vykreslen celý mrak skládající se z několika billboardů. Ještě efektivnějším řešením je vykreslit okolní mraky do např. osmi textur, které obklopí pozorovatele (viz. Obrázek 10).



Obrázek 10: Znázornění systému impostorů obklopujících pozorovatele. Převzato z [WAN04].

4.3 Krájení ploché „3D textury“

Další metodou pro zobrazení volumetrických dat, je tzv. „krájení“. Provede se řez, který se jako textura nanese na polygon a vykreslí se před pozorovatele. Takto lze objemová data „nakrájet“, a výsledky uložit za sebe do jedné dvourozměrné textury (viz. Obrázek 11). Tento způsob je využit v algoritmu popsaném v [DOM06]. Autor modeluje mraky podle [DOB00] (viz. 3.1.1), ale pro



Obrázek 11: Vznik ploché „3D textury“. Převzato z [DOM06].

zobrazení používá právě plochou „3D texturu“. Výhodou algoritmu je jeho jednoduchost a rychlost. Bohužel může vznikat obrazový artefakt mezi jednotlivými řezy.

4.4 Nasvícení oblaků

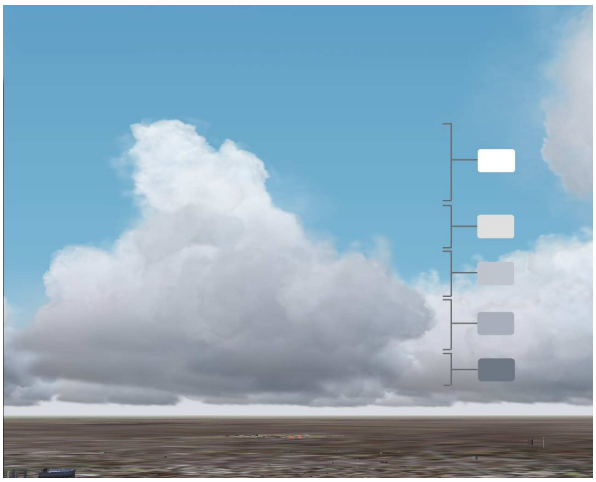
4.4.1 Jednoduchý a vícenásobný rozptyl světla

Osvětlovací modely založené na rozptylu světla simulují vyzařování, absorpci a rozptyl světla skrz objekt. Rozlišujeme metody založené na jednoduchém a vícenásobném rozptylu. Jednoduchý počítá osvětlení pouze jedním směrem, kterým je většinou směr pohledu pozorovatele. Tato metoda popsaná např. v [DOB00] je vhodná spíše k vykreslování kouře či dýmu. Pro oblaky je vhodnější, avšak výpočetně náročnější, využít vícenásobného rozptylu, který je přesnější (viz. Obrázek 12). Mark Harris v [HAR03] implementoval vícenásobný dopředný rozptyl. Jak ve svém textu popisuje, většina směrů nemá na výsledek vliv, a tak využil pouze dopředného rozptylu a rozptylu směrem k pozorovateli.

Obrázek 12: Rozdíl mezi jednoduchým (vpravo) a vícenásobným dopředným (vlevo) rozptylem světla. Převzato z [HAR02].

4.4.2 Předpřipravené nasvícení v modelu

Tvůrci hry Microsoft Flight Simulator 2004, kteří využívají předmodelovaných mraků začlenili nasvícení přímo do modelu. Oblak si rozdělili na pět částí, a každé přiřadili RGBA hodnotu (viz. Obrázek 13). Mezi těmito hodnotami je poté prováděna interpolace. Díky různým barvám a velikosti alfa složky je možné simulovat různé druhy mraků. Příkladem mohou být oblaky typu cumulonimbus, které mají tmavší spodní část, kdežto cumulus humilis zůstávají téměř bílé. V modelu mraku je také uložena transformace barvy při různém denním čase.

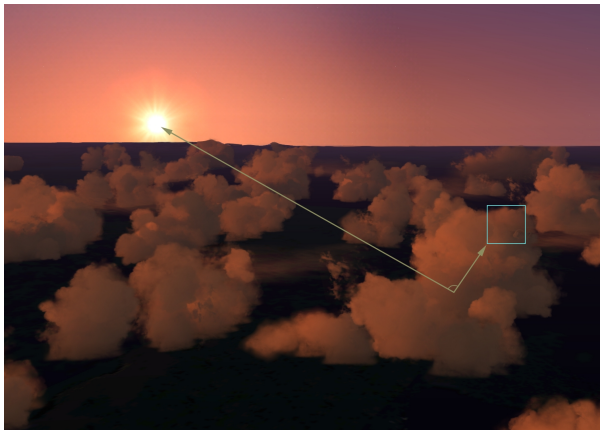


Obrázek 13: Nasvícení oblaku uložené v jeho modelu. Převzato z [WAN04].

Nasvícení od Slunce je opět řešeno s pomocí grafika. Ten při modelování rozdělí mrak na několik krychlí a pro výpočet nasvícení polygonu se využije skalární součin vzdálenosti polygonu od středu krychle, ve které se nachází, a vektoru od středu krychle ke Slunci, jak znázorňuje Obrázek 14. Pro celkový výpočet barvy vertexu je využit vztah:

$$C_{vertex} = (C_{amb} + C_{dir}) * C_{texture} * Alpha_{morph}$$

Vzhledem k tomu, že je většina potřebných informací uložena grafikem v modelu a část výpočtu je třeba realizovat na grafické kartě, je tento způsob nasvícení oblak nenáročný a lze jej provádět i na pomalejších počítačích (autoři algoritmu prováděli testy na procesoru s taktem 733MHz). Další podrobnosti tohoto řešení a testy jeho výkonnosti jsou popsány opět ve [WAN04].



Obrázek 14: Výpočet osvětlení od Slunce. Převzato z [WAN04].

5 Implementace algoritmu

5.1 Použité technologie a nástroje

Demonstrační aplikace je napsána v jazyce C++, kvůli jeho snadné přenositelnosti na různé platformy a poměrně vysoké efektivitě. Vývoj probíhal střídavě na operačním systému Fedora 10 (platforma x86-64) a Microsoft Windows Vista (x86). V linuxovém prostředí jsem využíval služeb IDE *Anjuta*, překladače GCC v. 4.3.2, debuggeru *gdb* a profileru *gprof*. V MS Windows byl veden vývoj v *Microsoft Visual Studio 2008 Professional*.

Neboť jsem vyvíjel na více platformách, zvolil jsem jako grafickou knihovnu OpenGL. Pro správu oken aplikace a její ovládání jsem využil knihovnu GLUT, která je taktéž multiplatformní. Protože jsou všechny modernější prvky grafických karet dostupné v OpenGL přes tzv. rozšíření (OpenGL Extensions), je vhodné použít některou z dostupných knihoven usnadňující přístup k těmto funkcím. Já jsem zvolil knihovnu GLEW ve verzi 1.5.1. Pro práci se shadery jsem zvolil jazyk vyvíjený společností nVidia nazvaný Cg, který na rozdíl od HLSL (DirectX) popř. GLSL (OpenGL) není vázaný na grafickou knihovnu.

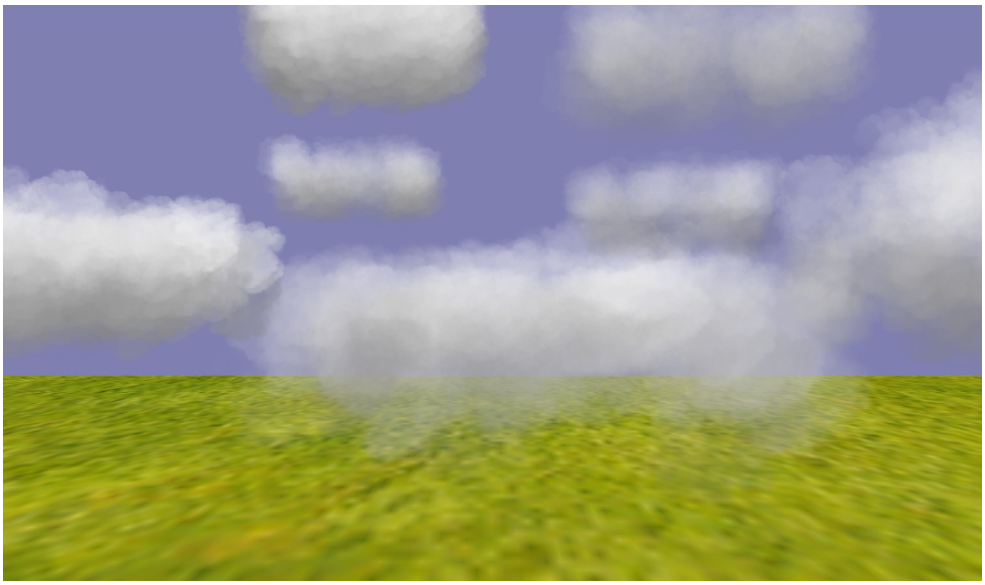
Pro spuštění přeloženého binárního souboru pro Microsoft Windows je třeba nainstalovat Microsoft Visual C++ 2008 Redistributable Package.

5.2 Modelování

Při své implementaci jsem vycházel z algoritmu popsaném v kapitole 3.3. Mým cílem však bylo nahradit práci grafika generátorem pseudonáhodných čísel.

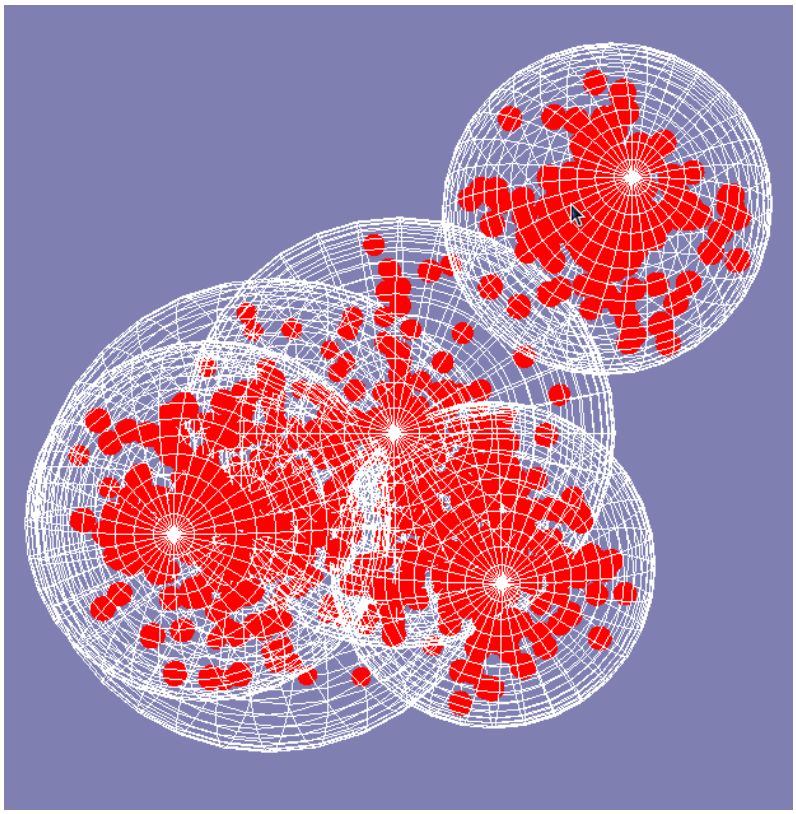
Princip mé implementace je vytvořit obalové těleso mraku, a do něj pak na pseudonáhodně vygenerované místo vložit polygony s texturou. Původně jsem částice umísťoval do kvádrů. Tato varianta nepřinesla uspokojivý výsledek, neboť mraky působily „krabicovitě“ (viz. Obrázek 15). Kvalitnějšího výsledku jsem dosáhl, když jsem kvádr nahradil několika koulemi. První koule je umístěna do středu mraku. Další je vygenerována na pozici v intervalu $\langle \frac{r}{3}, \frac{2r}{3} \rangle$ předchozí koule. Její poloměr je opět určován náhodně. Středy jednotlivých polygonů jsou generovány pomocí parametrické rovnice koule, kde $0 \leqslant \theta \leqslant \pi, -\pi < \varphi \leqslant \pi$.

$$x = x_0 + r \sin\theta \cos\varphi$$
$$y = y_0 + r \sin\theta \sin\varphi$$
$$z = z_0 + r \cos\theta$$



Obrázek 15: Výsledek modelování mraků do kvádrů.

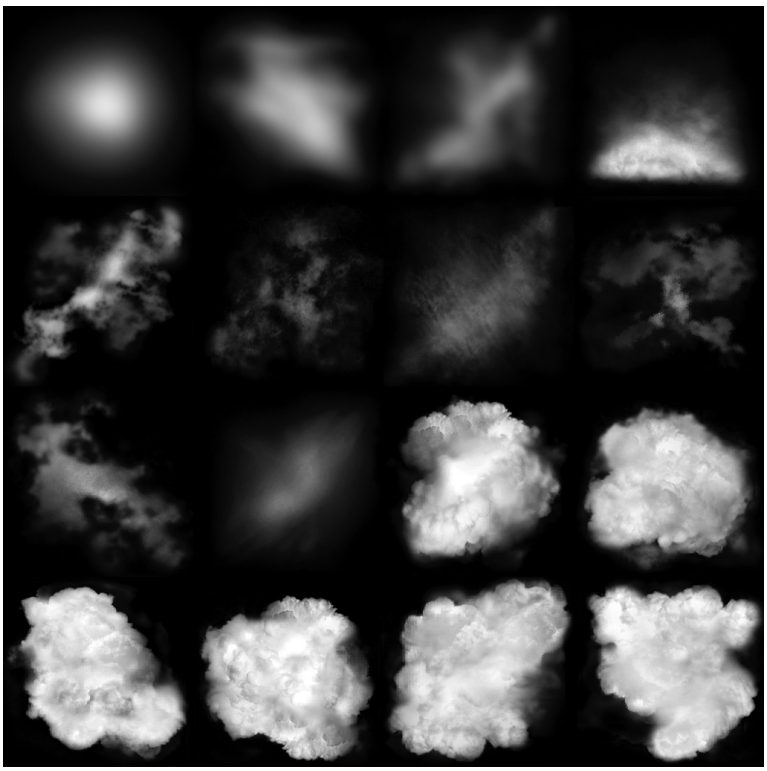
Výsledný model oblaku znázorňuje Obrázek 16 . V aplikaci je reprezentován jako kontejner standardní šablonové knihovny (STL) jazyka C++ (viz. např. [JOS05]), konkrétně *std::vector*, který obsahuje polohu středů všech polygonů v mraku.



Obrázek 16: Snímek obrazovky z demonstrační aplikace znázorňující vygenerovaný model mraku. Červené body znázorňují středy polygonů (billboardů).

5.3 Zobrazení

Jak jsem uvedl v kapitole (5.2) modely oblaků jsou uloženy v kontejneru *vector* knihovny STL. Stejně jako středy polygonů, jsou v jiném kontejneru uloženy i všechny jednotlivé mraky. Nejjednodušším zobrazením je tedy projít kontejner mraků, a pro každý vykreslit všechny jeho billboardy. Problémem je, že textury nanesené na polygonech jsou poloprůhledné, takže pro správné zobrazení musíme zajistit vykreslení dle jejich vzdálenosti od pozorovatele. To je vyřešeno pomocí malířova algoritmu viz. [ZAR04]. Před samotným zobrazením mraku jsou tedy polygony seřazeny s pomocí funkce standardní knihovny jazyka C++, pracující nad kontejnery, *std::sort*.



Obrázek 17: Katalog textur, převzatý z [WAN04].

Při generování mraků, je každému přidělena náhodná textura, která je reprezentována jako index do katalogu textur (viz. Obrázek 17). Ačkoliv jsem implementoval pouze mraky typu *cumulus* a využívám pouze textury z poslední řady, pro případné další rozšíření jsem ponechal všechny textury. Textura použitá v demonstrační aplikaci nemá černé pozadí, které je nahrazeno průhledným. Výsledek použitých algoritmů zastupuje Obrázek 18.

Obrázek 18: Snímek obrazovky demonstrační aplikace.

5.4 Stínování

Pro stínování jsem opět použil podobný způsob jako v algoritmu [WAN04], viz. 4.4.2. Práci grafika, jsem nahradil jednoduchým gradientem. Na vrcholu mraku má textura barvu bílé, a čím je polygon níže jeho „bělost“ postupně klesá. Výpočet lze vyjádřit vztahem:

$$barva = \frac{\frac{v}{2} + y}{v} + 0.5 \quad ,$$

kde *v* je výška mraku a *y* je vertikální složka pozice polygonu v mraku. Konstanta *0.5* zmírňuje černou na šedou, neboť černá na základně mraku nepůsobí realisticky. Vypočtená konstanta je uložena do struktury společně s pozicí středu a při vykreslování je předána fragment shaderu, kde je vynásobena s finální barvou polygonu.

5.5 Průhlednost

Způsob zobrazení, kdy obalové těleso obsahuje větší počet poloprůhledných polygonů, mi jednoduše zajistil další důležitou vlastnost, kterou by měl algoritmus pro vykreslování mraků mít. Tou je průlet samotným oblakem. Neboť jsou však všechny polygony stejně průhledné, nastává problém při „výletu“ z mraku, kdy poslední polygon je relativně neprůhledný, a není zde tedy žádný pozvolný

přechod. Situaci jsem vyřešil zavedením postupně přibývající průhlednosti se vzdáleností od středu mraku (výsledek viz. Obrázek 19).

Dalším využitím průhlednosti mraku je jeho formování a rozplynutí. Pro každý mrak je vygenerovaná celková průhlednost, která se sčítá s alfa složkou jednotlivých polygonů. Díky přibývající průhlednosti od středu se tedy mraky poměrně realisticky formují.

Celková alfa složka je opět předána fragment shaderu, který ji nastaví výslednému pixelu.

Obrázek 19: Snímek obrazovky demonstrační aplikace zachycující průlet mrakem.

5.6 Optimalizace

Vykreslování velkého množství polygonů je velmi náročné, a proto je vhodné zavádět nejrůznější optimalizace.

První volbou, která se přímo nabízí je využití impostorů (viz. 4.2.3). Výhodou tohoto řešení je, že se mrak vykreslí do textury pouze v nějakém intervalu (popř. při splnění určité podmínky), a není třeba vykreslovat všechny jeho billboardy pro každý snímek.

Pro tuto optimalizaci jsem využil rozšíření OpenGL nazvaného „framebuffer object“ (FBO), které umožňuje renderovat přímo do textury a data tedy není třeba kopírovat. Pro spuštění programu je nutné, aby grafická karta toto rozšíření podporovala.

5.7 Popis implementace

Aplikaci jsem rozčlenil do několika modulů. Snahou bylo oddělit modelování mraků od vykreslovacích a inicializačních procedur.

Třída zapouzdřující vyvinutý „mrakový“ systém se nazývá *CSky*. Je umístěna do souborů *sky.h* a *sky.cpp*. Konstruktoru této třídy se předává počet mraků jež má model obsahovat. Ty jsou poté rozmístěny do předem vymezených prostor, které jsou uloženy jako statické proměnné třídy.

Mraky jsou reprezentovány třídou *CCloud* v souborech *cloud.h* a *cloud.cpp*. Vzhled mraků ovlivňuje mnoho faktorů. Prvním je počet polygonů reprezentující mrak. Dále je to počet koulí, ze kterých se mrak skládá, jejich velikost a vzájemná vzdálenost. Změna kteréhokoliv z těchto parametrů koreluje s výkonem celého systému a samozřejmě s výsledným vzhledem. Lze tak například vygenerovat jiný typ oblaků.

Některé pomocné funkce a třídy jsou zahrnuty v souborech *aux_func.h* a *aux_func.cpp*. Je zde funkce na načítání textur ze souborů *tga* a třída pro reprezentaci obalových těles mraků potřebná při renderování do textury. Soubory point3d.cpp a point3d.h obsahují třídu pro práci s bodem v prostoru.

Reprezentace pozice kamery je umístěna v souboru *camera.cpp* a *camera.h*. Třída CCamera umožňuje nastavení směru pohledu a změnu pozice. Pro jednoduchou dostupnost v ostatních modulech a zaručení pouze jedné instance je třída implementována jako návrhový vzor singleton (viz. např. [ALE04]).

Funkce starající se o inicializaci a vykreslování jsou umístěny v souboru *main.cpp*. Vstupní funkcí do aplikace je jako ve všech C/C++ aplikacích funkce *main()*. Tato funkce vytváří okno, nastavuje callback funkce knihovny GLUT pro odchytávání stisknutých kláves a pohybu myši a nakonec invokuje hlavní smyčku zpráv. Při studiu práce s knihovnou GLUT mi velmi pomohl tutoriál [FER08], kterým jsem se inspiroval. Funkce *init()* volaná z funkce *main()* inicializuje systém mraků, načte textury ze souborů, zinicializuje Cg shadery a nastaví framebuffer object. Zavedení shaderů zahrnuje načtení i překlad zdrojových kódů a nastavení správného profilu podporovaného grafickou kartou systému. Pokud se profil nepodaří nastavit, je aplikace ukončena. Kód shaderů je uložen ve složce *Cg* a není příliš složitý. Vertex shader je v souboru *vs.cg* a víceméně pouze přeposílá data aplikace do fragment shaderu. Ten je uložen v souboru *fs.cg* a v prvním kroku se stará o nanesení textury. Výsledný pixel vynásobí gradientem (viz. 5.4) a nakonec nastaví průhlednost (viz. 5.5). Soubor *main.cpp* dále obsahuje především funkce pro vykreslování. Zejména je to *renderScene()*, která je volána při každém snímku a celé vykreslování řídí. Tato funkce každých 10 sekund vypisuje do konzole aktuální hodnotu FPS[1]. Dle potřeby volá další funkce pro renderování mraků, kterými jsou *renderCloud()*, *renderCloudImpostor()* a *renderCloudToTexture()*.

1 FPS (frames per second) je označení pro počet zobrazených snímků za vteřinu

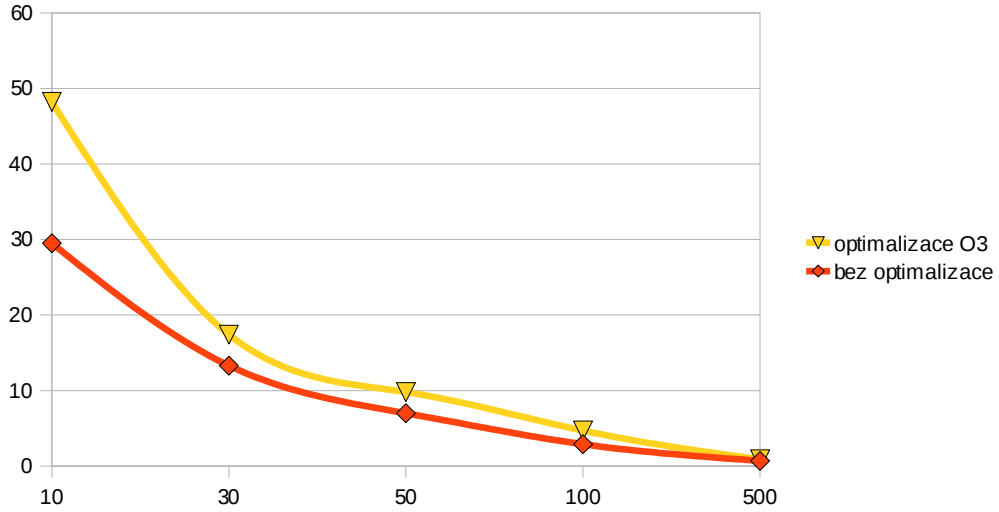
Kompletní popis je uveden v programové dokumentaci, která je přiložena na DVD (viz. příloha A).

5.8 Ovládání demonstrační aplikace

Ovládání pohybu ve scéně je umožněno pomocí šipek, rozhled scénou je nastaven na pohyb myši, jako ve většině počítačových her. Klávesou *F4* je možné zapnout/vypnout zobrazení pomocí impostorů. Klávesa *F1* umožňuje zobrazit scénu s pomocí drátových modelů (tzv. wireframe). Ukončení aplikace lze provést stiskem klávesy *Esc*.

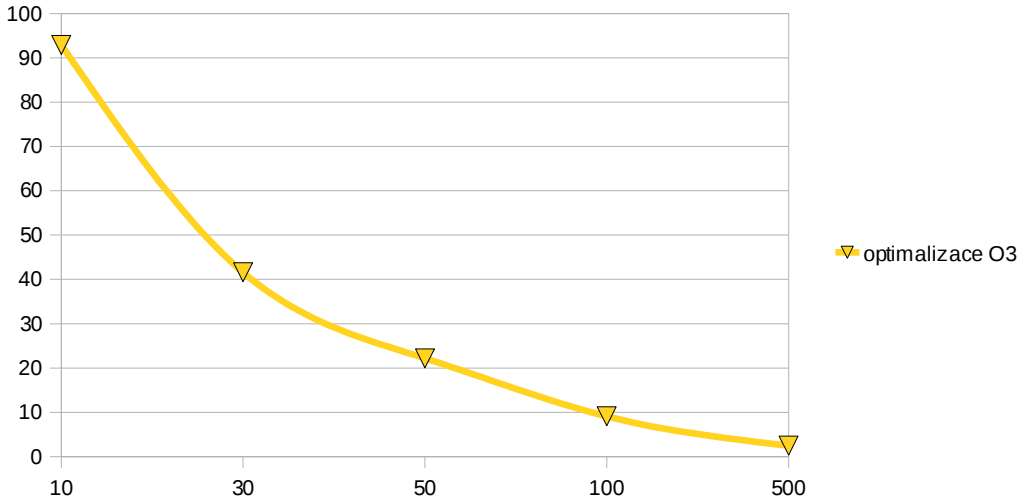
5.9 Testování a výkonnost

Měření výkonnosti systému je velmi obtížné, neboť mraky jsou generovány náhodně, takže při každém spuštění programu jsou jiné podmínky. Veškeré uvedené hodnoty jsou tedy spíše orientační.



Graf 1: Výkonnost systému při 40ti polygonech na jednu kouli reprezentující část mraku. Na ose x je znázorněn počet mraků ve scéně, na ose y počet snímků zobrazených za sekundu.

Během testování mě zaujalo, jak velký vliv má nastavení překladače na výkon. Verze přeložena s parametrem O3 je řádově o 30% rychlejší, než verze bez optimalizací překladače (viz. Graf 1).



Graf 2: Výkonnost systému při 15ti polygonech na jednu kouli reprezentující část mraku. Na ose x je znázorněn počet mraků ve scéně, na ose y počet snímků zobrazených za sekundu.

Výsledky testování nepřekvapí. Počet FPS je závislý na počtu mraků ve scéně a ty poté na počtu polygonů, ze kterých se skládají. Pozitivní je, že i 15 polygonů na modelovou kouli, tzn. že se mrak skládá ze 75 až 225ti polygonů, pro zobrazení mraku plně dostačuje (viz. Obrázek 20).

Obrázek 20: Snímek z demonstrační aplikace spuštěné s nastavením 15ti polygonů na modelovou kouli a 30 mraků na scéně.

Veškeré testy byly provedeny na notebooku Dell XPS M1330, s dvoujádrovým procesorem Intel Core 2 T8100@2GHz, 4GB operační paměti a grafickou kartou nVidia GeForce 8400GS v operačním systému Fedora 10 na platformě x86-64. Aplikace byla spuštěna v okně o rozměrech

640x640 pixelů. Testy jsem prováděl také v systému Microsoft Windows XP na počítači s procesorem Intel Core 2 a grafickou kartou ATI X1600. Počet zobrazených snímků byl na tomto stroji cirka dvojnásobně větší.

5.10 Problémy a komplikace

U víceprocesorových systémů v kombinaci s operačními systémy společnosti Microsoft a grafickými kartami nVidia jsem zaznamenal neoprávněné přístupy aplikace do paměti. Chyba je ve sdílené knihovně dodávané společností nVidia. Problém lze vyřešit nastavením afinity aplikace k jednomu z dostupných procesorů.

5.11 Možná rozšíření

Rozšířeních pro tuto práci je jistě celá řada. Vhodné by bylo zejména přidat více druhů oblaků, čehož by se dalo dosáhnout volbou jiného obalového tělesa, změnou textury, změnou stínování a průhlednosti. Dalším rozšířením by bylo přidat dynamiku mraků v závislosti na nastavené síle větru. Možnosti vylepšení vždy nabízí optimalizace, kde by se daly převést některé výpočty na procesor grafického akcelerátoru nebo implementovat, některý z algoritmů viditelnosti.

Zajímavým rozšířením by bylo implementovat algoritmus, který by generoval mraky podle aktuálních družicových snímků.

Přidání atmosferických efektů (déšť, sníh, duha, atd.), Slunce, či propracované zobrazení atmosféry samotné s vykreslováním mraků příliš nesouvisí, nicméně by scéně dodalo na realističnosti.

Pro experimentování s parametry by bylo velmi výhodné využít služby některého skriptovacího jazyka a do aplikace integrovat konzolové rozhraní pomocí něhož by se daly parametry měnit za běhu programu. Vhodným kandidátem pro tuto funkci by byl skriptovací jazyk *Lua*. Zajímavé články na toto téma jsou uvedeny ve čtvrté kapitole publikace [DIC05].

6 Závěr

Cílem této práce bylo popsat algoritmy schopné vymodelovat a zobrazit mraky v reálném čase, a některý algoritmus implementovat.

Při implementaci demonstrační aplikace jsem vycházel z algoritmu popsaném ve [WAN04], ve kterém jsem se ovšem snažil nahradit práci grafiků generátorem pseudonáhodných čísel. Esenciálním vybavením pro implementaci této metody zobrazení je znalost a orientace v reprezentaci modelů pomocí obrázků a bodů. Před samotnou implementací bylo třeba prostudovat pokročilejší techniky OpenGL jako práce s OpenGL Extensions, se shadery, či s renderováním do textury za pomocí FBO.

Výsledná aplikace umožňuje zobrazit mraky a pohybovat se v nich na dnes průměrně výkonných grafických akcelerátorech v reálném čase. Aplikace byla testována na operačních systémech Microsoft Windows a Linux.

Pokračování této práce by se mohlo ubírat ve snaze zobrazit aktuální počasí. Bylo by třeba implementovat podporu více druhů oblaků, atmosferické efekty a modely mraků generovat z aktuálních družicových snímků. Další možná rozšíření jsou uvedena v kapitole 5.11.

Literatura

[ALE04] ALEXANDRESCU, Andrei. Moderní programování v C++. 1. vyd. Brno: Computer Press, 2004. s. 340. ISBN 80-251-0370-6

[DIC05] DICKHEISER, Mike. Game Programming Gems 6. 2005. s. 695. ISBN 1-58450-450-1.

[DOB00] DOBASHI, Yoshinori, et al. A Simple, Efficient Method for Realistic Animation of Clouds . In: *SIGGRAPH '00: Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, New York: ACM Press/Addison-Wesley Publishing, 2000. s. 19-28. ISBN 1-58113-208-5

[DOM06] DOMAŃSKI, Dariusz. Fast algorithm for cloud rendering using flat „3D textures". In: *WSCG'2006 Posters Proceedings.* Plzeň: University of West Bohemia, 2006. s. 5-6. ISBN 80-86943-04-6

[FER03] FERNANDO, Randima – KILGARD, Mark J. *The Cg Tutorial: The Definitive Guide to Programmable Real-Time Graphics.* Addison-Wesley, 2003. s. 384. ISBN 0321194969

[FER08] FERNANDES, António Ramires. *OpenGL @ Lighthouse 3D – GLUT Tutorial* [online]. [cit. 10.9. 2008]. Dostupné z WWW: <http://www.lighthouse3d.com/opengl/glut/>

[GAM09] Gamespot [online]. [cit. 16.5.2009]. Dostupné z WWW: <http://www.gamespot.com>

[HAR02] HARRIS, Mark. Real Time Cloud Rendering [online]. [cit. 14.5.2009]. Dostupný z WWW: <http://www.markmark.net/clouds/>

[HAR03] HARRIS, Mark. *Real-Time Cloud Simulation and Rendering.* 2003.

[HAS05] HASAN, M.M. - SAZZAD KARIM, M. - AHMED, E. Generating and Rendering Procedural Clouds in Real Time on Programmable 3D Graphics Hardware. In: *9th International Multitopic Conference, IEEE INMIC 2005.* Karachi, 2005. s. 1-6. ISBN 0-7803-9429-1

[JOS05] JOSUTTIS, Nicolai M. *C++ Standardní knihovna a STL, Kompletní průvodce.* 1. vyd. Brno: Computer Press, 2005. s. 743. ISBN 80-251-0511-1

[MCR05] MCREYNOLDS, Tom – BLYTHE, David. Advanced Graphics Programming Using OpenGL. Amsterdam: Morgan Kaufmann, 2005. ISBN 1-55860-659-9

[MIY01] MIYAZAKI, Ryo, et al. A Method for Modeling Clouds based on Atmospheric Fluid Dynamics. In: Proceedings of the 9th Pacific Conference on Computer Graphics and Applications. Washington DC: IEEE Computer Society, 2001. ISBN 0-7695-1227-5

[NET84] NETOPIL, Rostislav, et al. *Fyzická geografie.* 1. vyd. Praha: SPN, 1984. 273 s.

[RAC08] RACKO, Stanislav. *Meteorologický atlas oblaků* [online]. 29.7. 2008 [cit. 2.5. 2009]. Dostupný z WWW: <http://www.chmi.cz/meteo/om/mk/atlasobl>

[ROT00] ROTH, Günter. *Encyklopedie počasí.* Praha: Euromedia group., 2000. 296 s. ISBN 80-242-0228-X

[TUT09] *Tutorials - FS2004 SDK* [online]. [cit. 4.5. 2009]. Dostupný z WWW: <http://www.scenery.org/tutorials_fs2k4_SDK.htm>

[WAN04] WANG, Niniane. Realistic and Fast Cloud Rendering. *Journal of graphics tools*, 2004, vol. 9, no. 3, s. 21-40.

[ZAR04] ŽÁRA, Jiří, et al. *Moderní počítačová grafika.* 2. vyd. Brno: Computer press, 2004. s. 609. ISBN 80-251-0454-0

Seznam příloh

Příloha A DVD obsahující zdrojové kódy demonstrační aplikace, potřebné knihovny a programovou dokumentaci